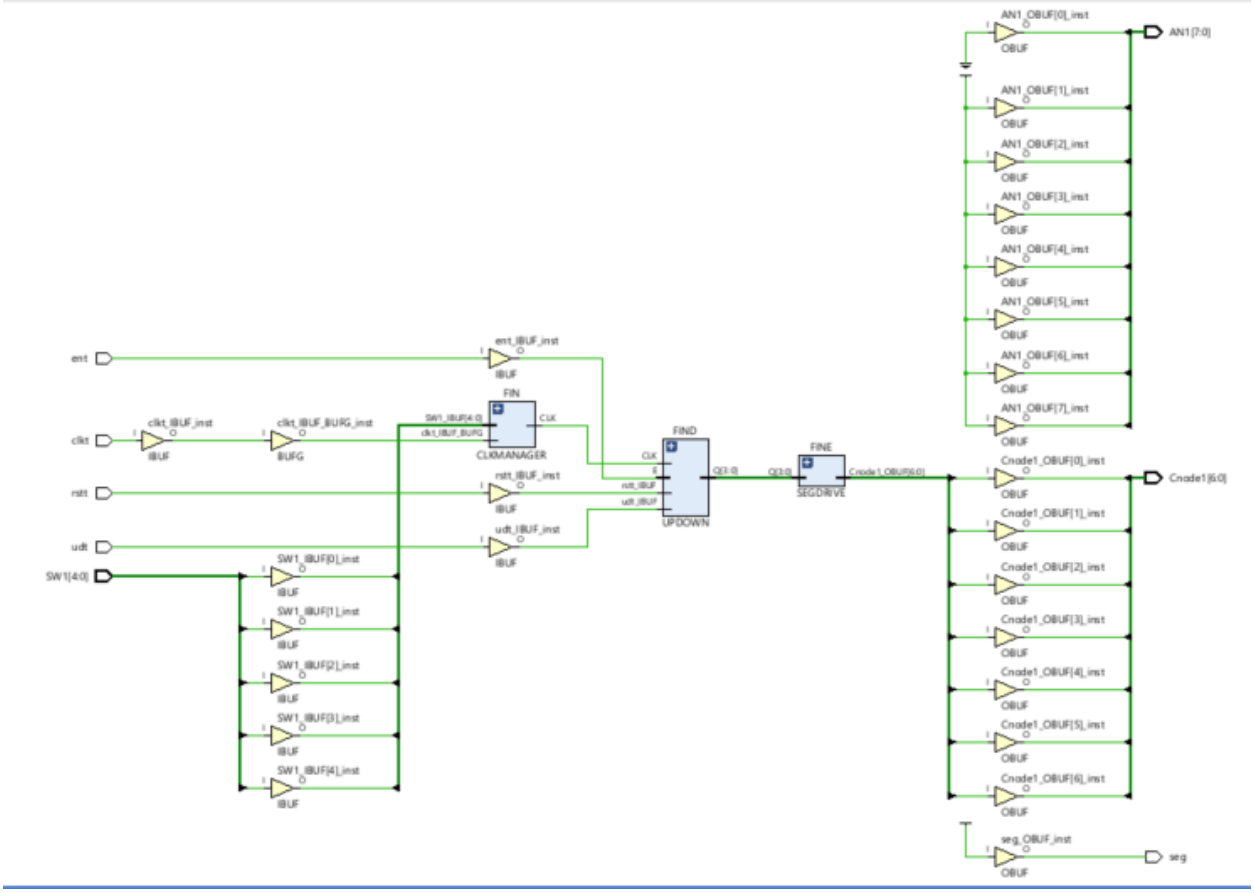
ECE3300L Lab3 Group H Report (Sherwin Sathish & Mohamed Hamida)

SCHEMATIC():



Xdc for top.v:

```
## This file is a general .xdc for the Nexys A7-100T
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clkt}];# IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clkt}];


##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { SW1[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { SW1[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
```

```
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { SW1[2] }];
#IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { SW1[3] }];
#IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { SW1[4] }];
#IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { udt }]; #IO_L7N_T1_D10_14
Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { ent }];
#IO_L17N_T2_A13_D29_14 Sch=sw[6]
#set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { ent }]; #IO_L5N_T0_D07_14
Sch=sw[7]
#set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { cinbit }]; #IO_L24N_T3_34
Sch=sw[8]
#set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { SW[9] }]; #IO_25_34 Sch=sw[9]
#set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { SW[10] }];
#IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
#set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { SW[11] }];
#IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports { SW[12] }]; #IO_L24P_T3_35
Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports { SW[13] }];
#IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports { SW[14] }];
#IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
#set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { SW[15] }];
#IO_L21P_T3_DQS_14 Sch=sw[15]

## LEDs
#set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { sumbit[0] }];
#IO_L18P_T2_A24_15 Sch=led[0]
#set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { sumbit[1] }];
#IO_L24P_T3_RS1_15 Sch=led[1]
#set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { sumbit[2] }];
#IO_L17N_T2_A25_15 Sch=led[2]
#set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { sumbit[3] }];
#IO_L8P_T1_D11_14 Sch=led[3]
#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { coutbit }]; #IO_L7P_T1_D09_14
Sch=led[4]
#set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { LED[5] }];
#IO_L18N_T2_A11_D27_14 Sch=led[5]
#set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { LED[6] }];
#IO_L17P_T2_A14_D30_14 Sch=led[6]
#set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { LED[7] }];
#IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { LED[8] }];
#IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { LED[9] }];
#IO_L14N_T2_SRCC_14 Sch=led[9]
#set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { LED[10] }];
#IO_L22P_T3_A05_D21_14 Sch=led[10]
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { LED[11] }];
#IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=led[11]
```

```
#set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { LED[12] }];
#IO_L16P_T2_CSI_B_14 Sch=led[12]
#set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { LED[13] }];
#IO_L22N_T3_A04_D20_14 Sch=led[13]
#set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { LED[14] }];
#IO_L20N_T3_A07_D23_14 Sch=led[14]
#set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { LED[15] }];
#IO_L21N_T3_DQS_A06_D22_14 Sch=led[15]

## RGB LEDs
#set_property -dict { PACKAGE_PIN R12   IOSTANDARD LVCMOS33 } [get_ports { LED16_B }];
#IO_L5P_T0_D06_14 Sch=led16_b
#set_property -dict { PACKAGE_PIN M16   IOSTANDARD LVCMOS33 } [get_ports { LED16_G }];
#IO_L10P_T1_D14_14 Sch=led16_g
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { LED16_R }];
#IO_L11P_T1_SRCC_14 Sch=led16_r
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { LED17_B }];
#IO_L15N_T2_DQS_ADV_B_15 Sch=led17_b
#set_property -dict { PACKAGE_PIN R11   IOSTANDARD LVCMOS33 } [get_ports { LED17_G }]; #IO_0_14
Sch=led17_g
#set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { LED17_R }];
#IO_L11N_T1_SRCC_14 Sch=led17_r

##7 segment display
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { Cnode1[6] }];
#IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { Cnode1[5] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { Cnode1[4] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { Cnode1[3] }];
#IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { Cnode1[2] }];
#IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { Cnode1[1] }];
#IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { Cnode1[0] }];
#IO_L4P_T0_D04_14 Sch=cg
set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports { seg }];
#IO_L19N_T3_A21_VREF_15 Sch=dp
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { AN1[0] }];
#IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { AN1[1] }];
#IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { AN1[2] }];
#IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { AN1[3] }]; #IO_L19P_T3_A22_15
Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { AN1[4] }]; #IO_L8N_T1_D12_14
Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { AN1[5] }];
#IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { AN1[6] }]; #IO_L23P_T3_35
Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { AN1[7] }];
#IO_L23N_T3_A02_D18_14 Sch=an[7]
```

```
##Buttons
#set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { rst }];
#IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { rstt }]; #IO_L9P_T1_DQS_14
Sch=btnc
#set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { dir }]; #IO_L4N_T0_D05_14
Sch=btnu
#set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { BTNL }];
#IO_L12P_T1_MRCC_14 Sch=btnl
#set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { BTNR }]; #IO_L10N_T1_D15_14
Sch=btnr
#set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { BTND }];
#IO_L9N_T1_DQS_D13_14 Sch=btnd
```

Code Detail:

`timescale 1ns / 1ps

```
`timescale 1ns / 1ps


module CLKMANAGER
(
   input clk,
   input rst,
   input [4:0] SW,
   output reg clkout =0
   );
   reg [31:0] sel;



   always@(posedge clk or posedge rst)
   begin: DREG
      if(rst)
         sel<= 32'd0;
      else
         sel<=sel+1;
   end

 always@(SW)
 begin


   case(SW)
         5'd0: clkout <= sel[0];
         5'd1: clkout <= sel[1];
         5'd2: clkout <= sel[2];
         5'd3: clkout <= sel[3];
         5'd4: clkout <= sel[4];
         5'd5: clkout <= sel[5];
         5'd6: clkout <= sel[6];
         5'd7: clkout <= sel[7];
         5'd8: clkout <= sel[8];
         5'd9: clkout <= sel[9];
         5'd10:clkout <= sel[10];
         5'd11:clkout <= sel[11];
```

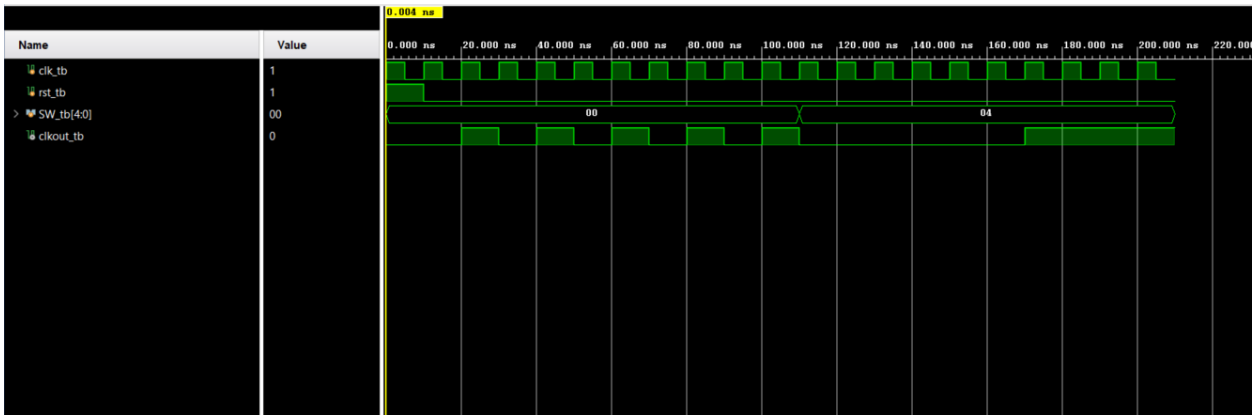
```
        5'd12:clkout <= sel[12];
        5'd13:clkout <= sel[13];
        5'd14:clkout <= sel[14];
        5'd15:clkout <= sel[15];
        5'd16:clkout <= sel[16];
        5'd17:clkout <= sel[17];
        5'd18:clkout <= sel[18];
        5'd19:clkout <= sel[19];
        5'd20:clkout <= sel[20];
        5'd21:clkout <= sel[21];
        5'd22:clkout <= sel[22];
        5'd23:clkout <= sel[23];
        5'd24:clkout <= sel[24];
        5'd25:clkout <= sel[25];
        5'd26:clkout <= sel[26];
        5'd27:clkout <= sel[27];
        5'd28:clkout <= sel[28];
        5'd29:clkout <= sel[29];
        5'd30:clkout <= sel[30];
        5'd31:clkout <= sel[31];

      endcase
   end


endmodule
```

The purpose of the clock manager is to have a counter and a 5-bit mux to output a clk with various levels of frequency depending on the integer div in this case. First we set up the counter by using an always loop that accounts for the reset, and sets up the different frequencies for all 32 values. Then we created an always loop that takes effect when the switches are changed, and we used a case statement for all 32 values of the 5-bit switches. Depending on the clkout that is selected by the switches, the frequency could vary for each clkout. We used the non-blocking assignment <= so that the code doesn't have to be done in order from top to bottom.

```
`timescale 1ns / 1ps


module CLKMANAGER_TB (

   );

   reg clk_tb,rst_tb;
   reg [4:0] SW_tb;
   wire clkout_tb;


   CLKMANAGER COMP(
   .clk(clk_tb),
   .rst(rst_tb),
   .SW(SW_tb),
   .clkout(clkout_tb)
   );

   initial
```

```
    begin
    clk_tb = 1;
    end

  always
    begin
    #5 clk_tb = ~clk_tb;
    //#20 SW_tb = SW_tb+1;
    end
  initial
    begin
      rst_tb = 1;
      SW_tb <= 0;
     #10
      rst_tb = 0;
     #100
      SW_tb = 4;
     #100
     $finish;
    end

endmodule
```

The purpose of this testbench is to test the code in the design source. We instantiate the variables of the design source code by calling that class,so that we can test them in this simulation source. We set the initial values for the inputs, and we use an always loop to make sure that the clock is toggling. We set a value for the switch to analyze the change in frequency.
EXAMPLE FOR CLKMANAGER:




```
`timescale 1ns / 1ps


module UPDOWN(
    input clk1,
    input rst1,
    input en,
    input ud,
```

```
output reg [3:0] tmp
);

always@(posedge clk1 or posedge rst1)
begin
if(en)

  begin
    if(rst1)
        begin
           if(ud)// 0 is up,  1 is down
           tmp<= 4'd9;
           else
           tmp<= 4'd0;
         end
    else
        begin
           if(ud)
              if(tmp==4'd0)
                 tmp <= 4'd9;
              else
                 tmp <= tmp-1;
           else
              if(tmp==4'd9)
                 tmp <= 4'd0;
              else
                 tmp <= tmp+1;
         end
        //tmp = tmp;
    end
    else
    tmp=0;

end



endmodule
```

The purpose of this code is to count up or down in BCD. The always@(posedge or rst) was used because this is a sequential circuit , and the non-blocking assignment <= is used as well because this code runs in parallel. The if statement with en is used first because without the en the updown counter wouldn't run. Next there is a if statement to see if the reset value is 1, if it is then we need to see whether the updown counter is 0 for up or 1 for down. This is because if we are counting from 9 to 0 then the value should reset to 9, whereas if we count from 0 to 9 it should reset to 0.

```
begin
            always@(posedge clk1 or posedge rst1)
   begin
   if(en)
```

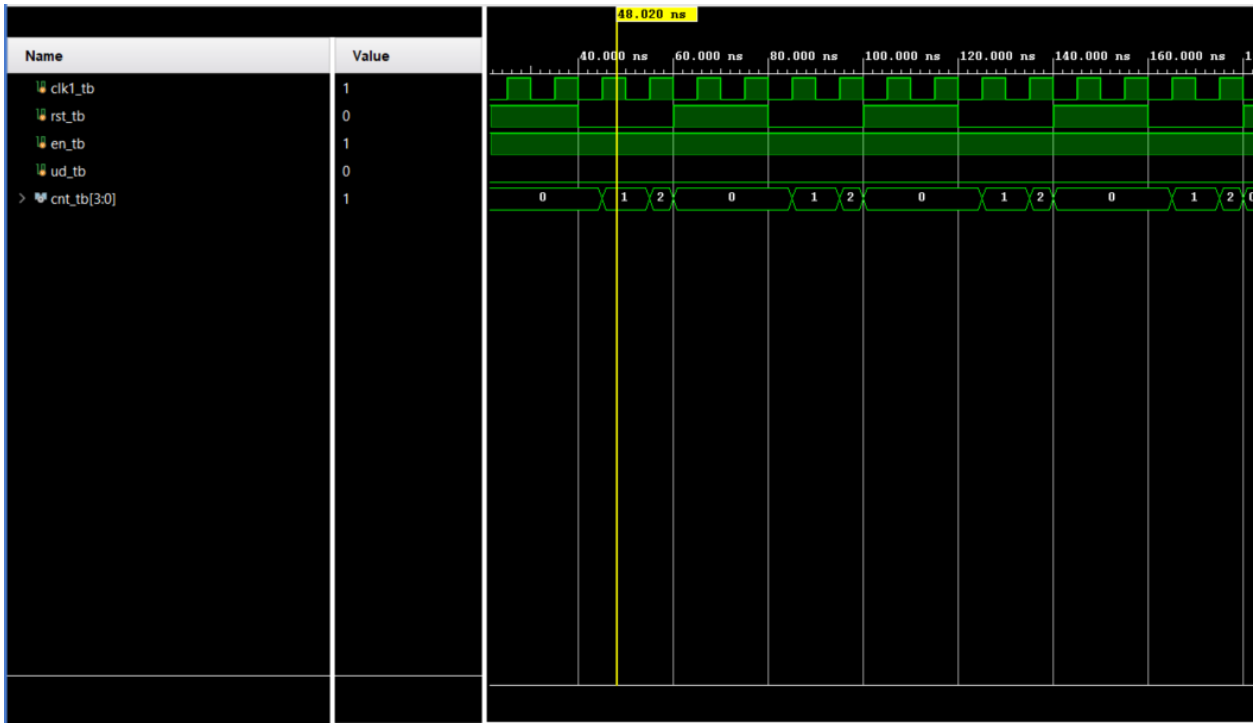
```
    begin
     if(rst1)
        begin
           if(ud)// 0 is up,  1 is down
           tmp<= 4'd9;
           else
           tmp<= 4'd0;
         end
      else
        begin
           if(ud)
              if(tmp==4'd0)
                 tmp <= 4'd9;
              else
                 tmp <= tmp-1;
           else
              if(tmp==4'd9)
                 tmp <= 4'd0;
              else
                 tmp <= tmp+1;
         end
        //tmp = tmp;
    end
    else
    tmp=0;

  end
```

The code above verifies the bcd because it checks whether the updown is 1 or 0, and based on what the maximum value for counting up and the minimum value for counting down is the tmp wire is reset to the beginning of the number counting scale. For example, the number counting scale is 0 to 9 for ud =0, so when it reaches 9 it will set back to 0 so that it doesn't go to 10 and so on. The tmp is the wire that connects the updown counter results to the 4-bit output that goes to the 7seg.

```
`timescale 1ns / 1ps
module UPDOWN_TB(
   );
   reg clk_tb,rst_tb;
   reg en_tb;
   reg ud_tb;
   wire [3:0] cnt_tb;
   UPDOWN COMP (
   .clk1(clk_tb),
   .rst1(rst_tb),
   .en(en_tb),
   .ud(ud_tb),
   .tmp(cnt_tb)
   );
   initial
      begin
      clk_tb = 0;
      rst_tb = 0;
      ud_tb = 0;
      en_tb = 1;
```

```
    end
  always
    begin
    #5 clk_tb = ~clk_tb;
    end
 always
    begin
    #20 rst_tb = ~rst_tb;
    end
endmodule
```

The purpose of this testbench is to test the code in the design source. We instantiate the variables of the design source code by calling that class,so that we can test them in this simulation source. We set the initial values for the inputs, then we use an always loop to make sure that the clock is toggling, and I had an always loop for reset just to test it out.
EXAMPLE FOR UPDOWN:




```
`timescale 1ns / 1ps

module SEGDRIVE(
  // input en,
   //input [31:0] SW,
   input [3:0] tmp_SW,
   output reg [6:0] Cnode,
   output dp,
   output wire [7:0] AN
   );
   //reg [31:0] temp;
```

```
    //reg [3:0] tmp_SW;

    assign AN = 8'b11111110;
    assign dp = 1'b1;
    always@(tmp_SW)
    begin
       case (tmp_SW)
          4'd0: Cnode <= 7'b0000001;
          4'd1: Cnode <= 7'b1001111;
          4'd2: Cnode <= 7'b0010010;
          4'd3: Cnode <= 7'b0000110;
          4'd4: Cnode <= 7'b1001100;
          4'd5: Cnode <= 7'b0100100;
          4'd6: Cnode <= 7'b0100000;
          4'd7: Cnode <= 7'b0001111;
          4'd8: Cnode <= 7'b0000000;
          4'd9: Cnode <= 7'b0000100;
          4'd10:Cnode <= 7'b0001000;
          4'd11:Cnode <= 7'b1100000;
          4'd12:Cnode <= 7'b0110001;
          4'd13:Cnode <= 7'b1000010;
          4'd14:Cnode <= 7'b0110000;
          4'd15:Cnode <= 7'b0111000;
       endcase
    end

      endmodule
`timescale 1ns / 1ps

module SEGDRIVE_TB(

    );
    reg [3:0] tmp_SW_tb;
    wire [6:0] Cnode_tb;
    wire dp_tb;
    wire [7:0] AN_tb;


    SEGDRIVE COMP (
    .tmp_SW(tmp_SW_tb),
    .Cnode(Cnode_tb),
    .dp(dp_tb),
    .AN(AN_tb)

    );

    initial
       begin
       //dp_tb = 1;
       tmp_SW_tb=0;
       #10
       tmp_SW_tb=4;
       #1000
       $finish;
```

```
        end



endmodule
```

This code takes an input of 4 switches and depending on when the switches change value, one of the 16 possible outputs will be selected to display on the 7 segment display. AN and dp are assigned in the beginning to make sure it is only one digit without the decimal place being shown. The different cases range from 0 to f, and the 4-bit switch value selects the specific case. In this particular code 0 represents on, while 1 represents off.

```
            `timescale 1ns / 1ps

module top(
    input clkt,
    input rstt,
    input ent,
    input [4:0] SW1,
    input udt,
    output seg,
    output [6:0] Cnode1,
    output [7:0] AN1
    );



    wire slowclk_out;
     CLKMANAGER FIN(
     .clk(clkt),
     //.rst(rstt),
     .SW(SW1),
     .clkout(slowclk_out)
    );


   wire [3:0] tmp_cnt;
     UPDOWN FIND(
     .rst1(rstt),
     .en(ent),
     .clk1(slowclk_out),
     .tmp(tmp_cnt),
     .ud(udt)
    );

 SEGDRIVE FINE(
   .tmp_SW(tmp_cnt),
   .Cnode(Cnode1),
```

```
  .dp(seg),
  .AN(AN1)
  );


endmodule
```

The top file basically instantiates the clock, switches, and clock out of the CLKMANAGER. It instantiates the reset, enable, clock, updown, and 4-bit output of the UPDOWN, and the SEGDRIVE switches, Cnode, dp, and Anode. Two wires are created to connect everything together. The slowclk_out wire connects the output clk of the clk manager to the input of the updown counter clk. The wire tmp_cnt connects the 4-bit output of the updown to the input of the switches, so the values of 0 to 9 can be displayed on the FPGA board.

This code represents the following schematic of a clkmanager that generates 32 random frequencies and then sends it out as the clock of the updown which counts 0 to 9 or 9 to 0 at different frequencies. The 4-bit output represents the BCD value, which will be connected to the switches of the 7-segment display, so that it could display the BCD on the 7-segment display.

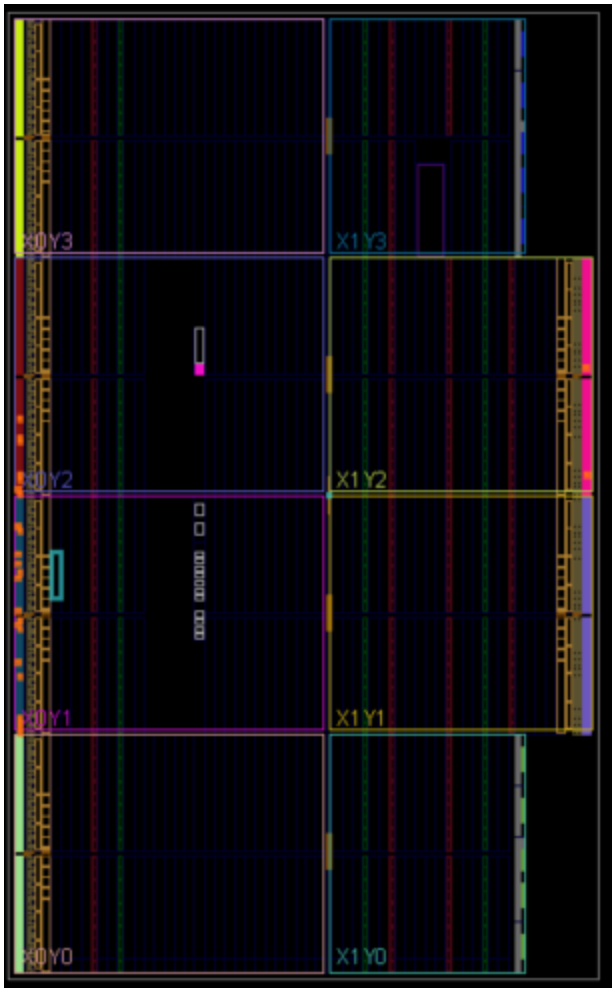
Corner Cases/Error :

One corner case can be the CLKMANAGER Testbench. At first the testbench wasn't working because of :

```
always@(SW)
 begin
```

The problem is that the loop can only run if the SW changes, so we had to change SW to posedge clk or posedge rst, so that the code could effectively work in the testbench.

IMPLEMENTED DESIGN/ TIMING SUMMARY:



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.005 ns	Worst Hold Slack (WHS):	0.336 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	32	Total Number of Endpoints:	32	Total Number of Endpoints:	33

All user specified timing constraints are met.

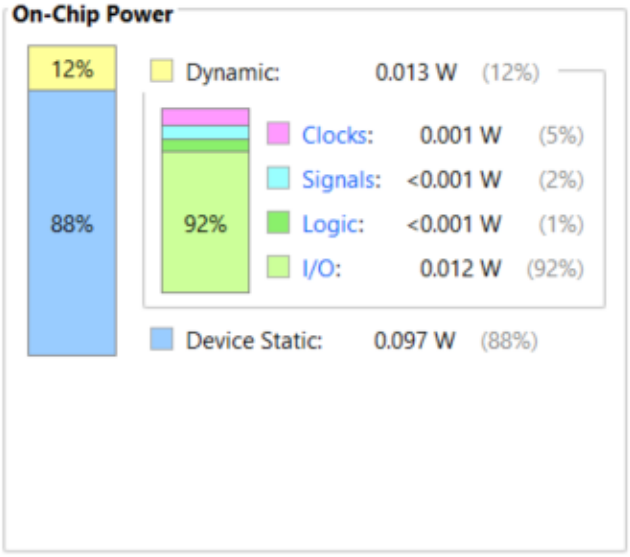
POWER SUMMARY:

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.11 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.5°C**
Thermal Margin:	59.5°C (12.9 W)
Effective ΘJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



UTILIZATION:

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)	BUFGCTRL (32)
N top	19	36	4	16	19	25	1
FIN (CLKMANAGER)	10	32	4	12	10	0	0
FIND (UPDOWN)	5	4	0	3	5	0	0
FINE (SEGDRIVE)	4	0	0	3	4	0	0

RESOURCE USAGE:

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
synth_1	constrs_1	Synthesis Out-of-date									19	36	0	0	0
impl_1	constrs_1	Implementation Out-of-date	7.005	0.000	0.336	0.000		0.000	0.110	0	19	36	0	0	0